O'REILLY®

Helion

Generatywna AI dla programistów

Praktyczne techniki i narzędzia

Sergio Pereira

Tytuł oryginału: Generative AI for Software Development:
Building Software Faster and More Effectively

Tłumaczenie: Radosław Meryk

ISBN: 978-83-289-3498-6

Authorized Polish translation of the English edition of *Generative AI for Software Development*
ISBN 9781098162276 © 2025 Goalstat Lda

This translation is published and sold by permission of O'Reilly Media, Inc.,
which owns or controls all rights to publish and sell the same.

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/genaip
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

ROZDZIAŁ 1.

Generowanie kodu i autouzupełnianie

Sztuczna inteligencja może w istotny sposób zwiększyć produktywność i kreatywność w procesie generowania kodu i jego autouzupełniania. W tym rozdziale przyglądam się, jak narzędzia oparte na AI redefiniują pracę programisty, przekształcając czasochłonny, ręczny proces w interaktywną, wydajną i mniej podatną na błędy pracę.

Wejście AI do świata generowania kodu nie polega jedynie na przyspieszeniu pisania. Istotą tej zmiany jest zdolność do rozumienia kontekstu pracy programisty, proponowania trafnych fragmentów kodu oraz tworzenia złożonych bloków przy minimalnym wkładzie użytkownika. Narzędzia te, oparte na zaawansowanych algorytmach uczenia maszynowego, korzystają z ogromnych repozytoriów kodu — zarówno publicznych, jak i prywatnych — co pozwala im stale udoskonalać swoje sugestie i zwiększać precyzję działania.

W tym rozdziale wyjaśnię, jak zmienia się rola inżyniera oprogramowania: dawniej polegała na samodzielnym wykonywaniu całego zadania, a dziś coraz częściej sprowadza się do oceny rezultatów generowanych przez narzędzia AI. Wymaga to precyzyjnego określania oczekiwań wobec tych narzędzi oraz dokładnej weryfikacji ich wyników, aby produkt końcowy spełniał wszystkie wymagania.

Narzędzia AI mają rozbudowane, imponujące możliwości, ale łatwo wpaść w pułapkę bezrefleksyjnego korzystania z ich wyników — na przykład wielu programistów decyduje się na otwieranie żądań pull request lub wdrażanie kodu na produkcję bez sprawdzenia, jak działa i dlaczego działa w określony sposób. Takie lekkomyślne podejście wiąże się z dwoma poważnymi zagrożeniami:

Przestarzały kod

Większość narzędzi AI jest szkolona na danych, które zdążyły się już zdezaktualizować. W efekcie mogą one sugerować przestarzałe frameworki lub funkcjonalności.

Błędne odpowiedzi

Duże modele językowe (ang. *large language models* — LLM), technologia stojąca za wszystkimi tymi narzędziami, potrafią generować tzw. halucynacje. Oznacza to, że w wynikach mogą pojawiać się fałszywe informacje, błędy lub funkcje kodu czy punkty końcowe API, które w rzeczywistości nie istnieją.

Inżynierowie oprogramowania i programiści powinni korzystać z narzędzi AI, aby pracować szybciej i efektywniej, ale nie po to, by zastąpić własny osąd. Podobnie jak w przypadku funkcji autouzupełniania, która stała się standardem w większości IDE, klawisz *Tab* potrafi oszczędzić mnóstwo czasu — ale sugestie bywają zarówno trafne, jak i całkowicie nietrafione. To od Twojej oceny zależy, czy je przyjąć, czy odrzucić.

Narzędzia AI omawiane w tym rozdziale wymagają takiego samego, stałego krytycznego podejścia. Często kod, który generują, działa bez zarzutu i idealnie odpowiada wymaganiom zadania. Innym razem jest niepełny, obarczony błędami, problemami z wydajnością lub innymi wadami wymagającymi poprawek. To Twoim zadaniem jest podjęcie decyzji, czy taki kod wykorzystać, odrzucić, czy poprawić.

Rodzaje narzędzi do generowania kodu

Narzędzia AI omawiane w tym rozdziale dzielą się na dwie główne kategorie, różniące się sposobem wykorzystania w procesie tworzenia oprogramowania:

Narzędzia przeglądarkowe

Do tej grupy należy m.in. ChatGPT. Wystarczy zalogować się w przeglądarce i pracować bezpośrednio w interfejsie online. Cała interakcja odbywa się przez internet, a Twój komputer nie wykonuje żadnych obliczeń. Takie narzędzia są proste w obsłudze i uniwersalne, ale ich największą wadą jest ograniczone okno kontekstu. Za każdym razem musisz ręcznie wprowadzać lub wklejać potrzebny fragment kodu bądź dokumentacji, co staje się szczególnie uciążliwe przy dużych projektach.

Narzędzia zintegrowane z IDE

Narzędzia takie jak GitHub Copilot działają jako wtyczki instalowane w środowisku programistycznym na Twoim komputerze. Po instalacji stają się integralną częścią procesu tworzenia oprogramowania — w tym samym miejscu, w którym piszesz kod. Ich największym atutem jest szerokie okno kontekstu: mogą uwzględnić całą bazę kodu jako punkt odniesienia w każdej interakcji.

Zastosowania

Miliony programistów korzystają z narzędzi opartych na sztucznej inteligencji, aby wspierać swoje codzienne zadania. Pięć najważniejszych obszarów, w których te narzędzia realnie wpływają na proces tworzenia oprogramowania, to:

Generowanie fragmentów kodu

Zamiast wpisywać ręcznie każdą funkcję i każdą linię kodu, wystarczy określić wymagania, które dany fragment ma spełniać. Narzędzie AI generuje następnie gotowy kod w jednym z najpopularniejszych języków programowania, takich jak Java, Python, PHP czy JavaScript. Dzięki temu można znacząco przyspieszyć zarówno prototypowanie, jak i cały proces tworzenia oprogramowania. Narzędzia opisane w tym rozdziale potrafią generować kod dla wielu typów zastosowań — od aplikacji webowych, przez analizę danych i skrypty

automatyzujące, po aplikacje mobilne. To przykład sytuacji, w której AI pomaga wypełnić lukę między pomysłem a jego realizacją, czyniąc proces rozwoju technologii szybszym i bardziej dostępnym.

Debugowanie

Debugowanie bywa czasochłonne i frustrujące, dlatego tu właśnie narzędzia AI okazują się niezwykle pomocne. Analizują komunikaty o błędach i problematyczne fragmenty kodu, a następnie proponują konkretne poprawki. Dzięki temu oszczędzasz czas, a jednocześnie rozwijasz własne umiejętności debugowania. Niektóre narzędzia (np. ChatGPT) potrafią dodatkowo wyjaśnić *przyczyny* błędów, a nawet wskazać kompromisy architektoniczne związane z ich unikaniem. Ta możliwość głębszego zrozumienia typowych problemów w programowaniu jest jednym z głównych powodów, dla których tak wielu programistów traktuje AI jako codziennego asystenta.

Przyspieszanie nauki

Narzędzia AI mogą pełnić rolę nauczyciela, gdy uczysz się nowego języka programowania, frameworka albo chcesz zrozumieć konkretne detale implementacji — np. jak dodać indeks do tabeli w MySQL czy pobrać dane transakcji z ostatniego miesiąca z API Stripe. Potrafią dostarczyć samouczków, przykładów i zwięzłych podsumowań dokumentacji w szerokim zakresie technologii. Taka interakcja edukacyjna pozwala znacznie przyspieszyć rozwój, niezależnie od tego, jakiego obszaru dotyczy nauka.

Optymalizacja kodu

AI wspomaga również inżynierów w przeglądaniu i optymalizowaniu kodu, co czyni ten proces bardziej wydajnym, czytelnym i łatwiejszym. Może podpowiadać refaktoryzację, wskazywać lepsze algorytmy czy sugerować praktyki zwiększające wydajność i bezpieczeństwo. Optymalizacja to zadanie, które wielu programistów odkłada na później, a z biegiem czasu drobne zaniedbania prowadzą do narastania długu technicznego, którego usunięcie staje się kosztownym przedsięwzięciem. Regularne korzystanie z narzędzi AI do przeglądów kodu na poziomie bieżących zadań może znacząco poprawić jakość całej bazy kodu.

Automatyzacja dokumentacji

Dokumentacja jest niezbędna do utrzymania i zrozumienia projektów programistycznych, jednak programiści często pomijają ją lub nadają jej zbyt niski priorytet. Niektóre narzędzia AI potrafią automatycznie tworzyć dokumentację — od komentarzy w kodzie, po opisy funkcji, klas i modułów. Pozwala to zaoszczędzić czas i zapewnia, że dokumentacja jest na bieżąco aktualizowana wraz z rozwojem kodu. Przejrzyste i kompletne materiały zwiększają czytelność kodu i ułatwiają współpracę w zespołach. Jest to szczególnie ważne w dużych projektach oraz w środowiskach open source, gdzie dobra dokumentacja stanowi warunek skutecznego udziału w projekcie innych programistów. Automatyzacja tego procesu poprawia też możliwości utrzymywania projektów i wspomaga efektywny transfer wiedzy w zespołach programistycznych.

Jak wspomniano wcześniej, narzędzia AI do tworzenia oprogramowania można podzielić na dwie główne grupy: działające w przeglądarce oraz zintegrowane ze środowiskiem IDE. Zacznijmy od narzędzi przeglądarkowych.

Narzędzia przeglądarkowe

AI dostępna w przeglądarce działa bezpośrednio po otwarciu strony internetowej, co czyni ją wyjątkowo wygodną w użyciu. Jej wadą jest jednak konieczność samodzielnego przekazywania całego kontekstu w prompcie. Przy pracy z dużymi bazami kodu czy złożonymi aplikacjami oznacza to konieczność wielu powtarzalnych operacji kopiuj – wklej między narzędziem AI, a środowiskiem IDE, w którym rozwijany jest projekt.

ChatGPT

ChatGPT (*https://chat.openai.com*) to model językowy opracowany przez OpenAI i prawdopodobnie najczęściej wykorzystywane narzędzie AI, które omawiamy w tej książce. Od premiery w listopadzie 2022 roku jego popularność rosła w zawrotnym tempie. W kwietniu 2025 roku liczba aktywnych użytkowników w ciągu tygodnia wyniosła około 800 milionów, co oznacza podwojenie bazy w zaledwie kilka tygodni. Platforma obsługuje ponad miliard zapytań dziennie i należy dziś do pięciu najczęściej odwiedzanych serwisów internetowych na świecie[1].

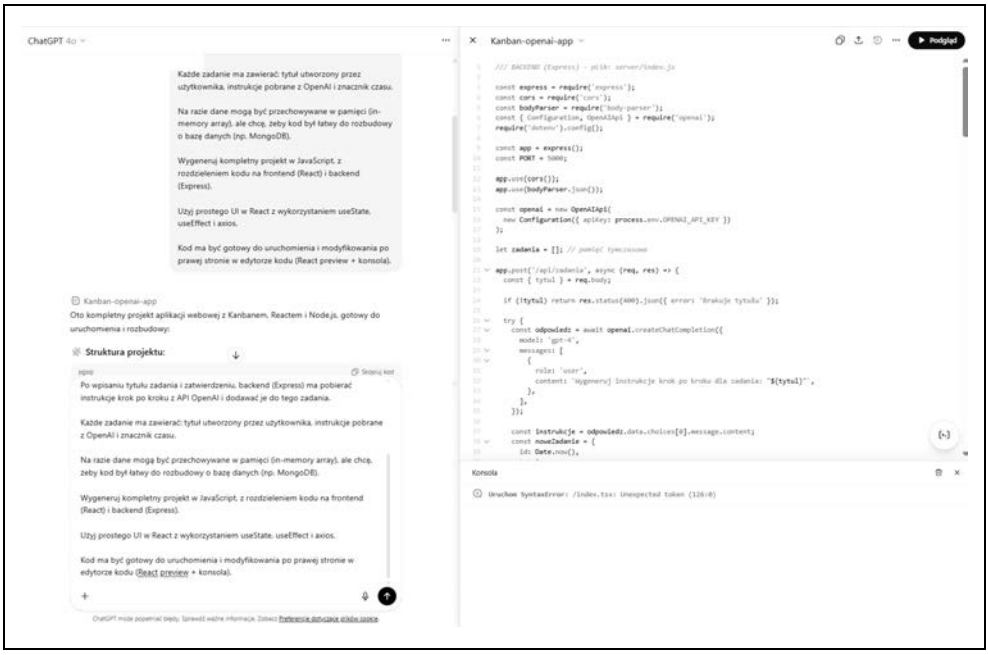
Popularność ChatGPT bezpośrednio przełożyła się na gwałtowny wzrost przychodów OpenAI — prognozy wskazują, że w 2025 roku przekroczą one 12,7 miliarda dolarów, wobec 3,7 miliarda w roku 2024. Źródłem tego wzrostu jest ponad 20 milionów subskrybentów planów Plus, Team i Pro, generujących co najmniej 415 milionów dolarów przychodu miesięcznie. Tak szybkie tempo upowszechnienia się tej technologii wynika z nieustannego wprowadzania innowacji. W maju 2024 roku zaprezentowano model GPT-4o, który wprowadził możliwości multimodalne, umożliwiając interakcję w czasie rzeczywistym za pomocą tekstu, głosu i obrazu. Kolejne aktualizacje — takie jak natywne generowanie obrazów oraz bardziej zaawansowane modele rozumowania, m.in. GPT-4.1 i o3 — dodatkowo poszerzyły zakres jego możliwości[2].

Ewolucja ChatGPT przekształciła go z prostego chatbota tekstowego w wszechstronnego asystenta AI, co całkowicie zmieniło sposób, w jaki indywidualni użytkownicy i firmy wchodzą w interakcję z technologią. Przyjazny interfejs i rozbudowane możliwości sprawiły, że stał się narzędziem nieodzownym — od zastosowań związanych z osobistą produktywnością, po rozwiązania dla przedsiębiorstw.

Jak pokazano na rysunku 1.1, ChatGPT udostępnia interfejs chatbota, w którym użytkownicy wpisują polecenia i w ciągu kilku sekund otrzymują odpowiedzi. Niedawno OpenAI wprowadziła edytor kodu, który można otworzyć po prawej stronie ekranu, wraz z konsolą i podglądem.

[1] Kerem Gûlen, *ChatGPT Just Hit 1 Billion Users and Melted Its Own Servers*, Dataconomy, 15 kwietnia 2025; *https://oreil.ly/yixrG*.

[2] Stephanie Palazzolo, Amir Efrati. *ChatGPT Revenue Surges 30%—in Just Three Months*, The Information, 1 kwietnia 2025; *https://oreil.ly/JMHJK*.



Rysunek 1.1. Interfejs użytkownika ChatGPT

To krok OpenAI w kierunku zmniejszenia dystansu między narzędziami działającymi w przeglądarce a rozwiązaniami zintegrowanymi z IDE. Dzięki możliwości edytowania i uruchamiania kodu bezpośrednio w ChatGPT ograniczono konieczność ciągłego kopiowania i wklejania między aplikacją a środowiskiem programistycznym.

Google Gemini

Gemini (*https://gemini.google.com*) to bezpośredni konkurent ChatGPT stworzony przez Google. Zadebiutował w grudniu 2023 roku jako następca Barda. Integruje się płynnie z ekosystemem Google, dzięki czemu poprawia komfort pracy użytkowników w takich aplikacjach jak Gmail, Dokumenty czy Arkusze. Do kwietnia 2025 roku Gemini zgromadził znaczącą bazę użytkowników — około 275 milionów aktywnych użytkowników miesięcznie[3].

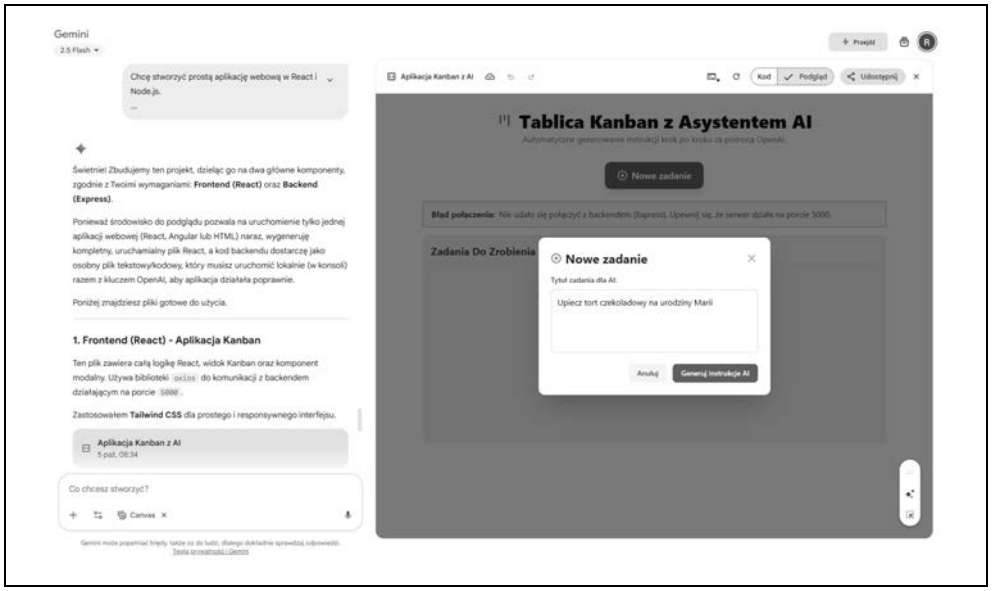
Możliwości platformy rozszerzyły się dzięki wprowadzeniu takich funkcji jak Gemini Live, która umożliwia prowadzenie rozmów w czasie rzeczywistym, oraz Audio Overviews, zamieniającej dokumenty w podsumowania w formie podcastów. Użytkownicy Gemini Advanced mogą także tworzyć krótkie filmy na podstawie poleceń tekstowych, co pozwala przygotowywać treści bez użycia tradycyjnych narzędzi do produkcji wideo.

Rozwój Gemini wspomaga ponad 1,5 miliona programistów, którzy z jego wykorzystaniem budują własne rozwiązania, co przyczynia się do różnorodności zastosowań. Jako kluczowy

[3] Sentisight.ai, *Google Gemini: How Has It Been Received by Users so Far?*, 24 stycznia 2025; *https://oreil.ly/jenAT*.

element strategii Google w obszarze AI, Gemini stale ewoluuje — oferuje innowacyjne rozwiązania dostosowane do szerokich potrzeb użytkowników.

Podobnie jak ChatGPT, Gemini działa w formie interfejsu czatu, w którym użytkownik wpisuje polecenia i otrzymuje odpowiedzi. Dodatkowo wprowadzono środowisko programistyczne w przeglądarce, z panelem kodu i podglądu po prawej stronie ekranu (rysunek 1.2). Ta funkcja sprawia, że dla wielu inżynierów oprogramowania Gemini staje się wystarczającym narzędziem do tworzenia niewielkich skryptów i projektów.



Rysunek 1.2. Interfejs użytkownika Google Gemini

Narzędzia zintegrowane z IDE

Przyjrzyjmy się teraz najważniejszym narzędziom opartym na środowiskach programistycznych, które wspierają pracę inżynierów oprogramowania. Obejmują one zarówno natywne IDE wyposażone w sztuczną inteligencję, jak i wtyczki AI do popularnych środowisk.

GitHub Copilot

GitHub Copilot (*https://oreil.ly/6tmUG*) został opublikowany w 2021 roku i szybko stał się jednym z kluczowych narzędzi w pracy programistów. Oferuje sugestie kodu generowane przez AI i integruje się z wieloma środowiskami programistycznymi. Do 2025 roku liczba płatnych subskrybentów przekroczyła milion, a z narzędzia korzystało ponad 77 tysięcy organizacji, co oznacza wzrost wykorzystania w przedsiębiorstwach rok do roku o 180%. Copilot stał się także ważnym filarem wzrostu finansowego GitHuba: przyczynił się do ponad 40-procentowego

wzrostu przychodów platformy, które w kwietniu 2025 roku osiągnęły roczną stopę przychodów na poziomie 2 miliardów dolarów[4].

Funkcjonalność Copilota wykracza daleko poza podstawowe uzupełnianie kodu. Obejmuje Copilot Chat, który umożliwia rozmowę z AI w celu uzyskania wyjaśnień i sugestii dotyczących kodu; Copilot Extensions, czyli integrację z narzędziami takimi jak Azure, Docker i MongoDB; oraz Copilot Pro+, który zapewnia dostęp do zaawansowanych modeli AI, w tym Claude 3.7 od Anthropic oraz Gemini Flash 2.0 od Google, co dodatkowo zwiększa wszechstronność narzędzia.

Wpływ Copilota na produktywność programistów jest znaczący. Badania pokazują, że osoby korzystające z tego narzędzia osiągają nawet 55-procentowy wzrost efektywności pracy z kodem i deklarują większe zadowolenie z jej wykonywania[5]. Dzięki stałym innowacjom i rosnącej liczbie użytkowników GitHub Copilot zmienia sposób tworzenia oprogramowania na całym świecie. Kodowanie staje się przez to bardziej dostępne i skuteczne.

Interfejs Copilota nie zmienia domyślnego wyglądu środowiska IDE, w którym jest zainstalowany. Dodaje jednak dodatkową warstwę skrótów klawiaturowych, które umożliwiają uruchomienie czatu — w formie panelu po prawej stronie ekranu (jak widać na rysunku 1.3) albo bezpośrednio w widoku kodu, co umożliwia autouzupełnianie lub interakcję z wybranymi fragmentami kodu.

Rysunek 1.3. Interfejs GitHub Copilota w IDE

GitHub Copilot integruje się ze wszystkimi popularnymi środowiskami programistycznymi, takimi jak Visual Studio Code, JetBrains, Eclipse czy Xcode. Dzięki temu rozwój narzędzia od samego początku przebiegał płynnie — większość programistów już korzystała z tych IDE, więc instalacja GitHub Copilota wymagała tylko jednego kliknięcia w panelu wyszukiwarki rozszerzeń.

[4] Wade Tyler Millward, *Microsoft Q4 2024: CEO Nadella Calls Copilot Growth Rate Fastest for Any M365 Suite*, CRN, 31 lipca 2024; *https://oreil.ly/JSCNJ*.

[5] Zasoby GitHub Copilota, *Measuring the Impact of GitHub Copilot*, GitHub [dostęp: 04 czerwca 2025]; *https://oreil.ly/85G7H*.

Podejście do wprowadzania asystentów AI na rynek zostało jednak szybko zakwestionowane przez środowiska IDE natywnie wspierające sztuczną inteligencję, do których wrócę za chwilę. Warto też zauważyć, że o ile Copilot początkowo oferował wyłącznie modele OpenAI, to wraz z rosnącą popularnością narzędzi takich jak Kursor i Windsurf umożliwia dziś wybór spośród modeli OpenAI, Anthropic i Google.

Cursor

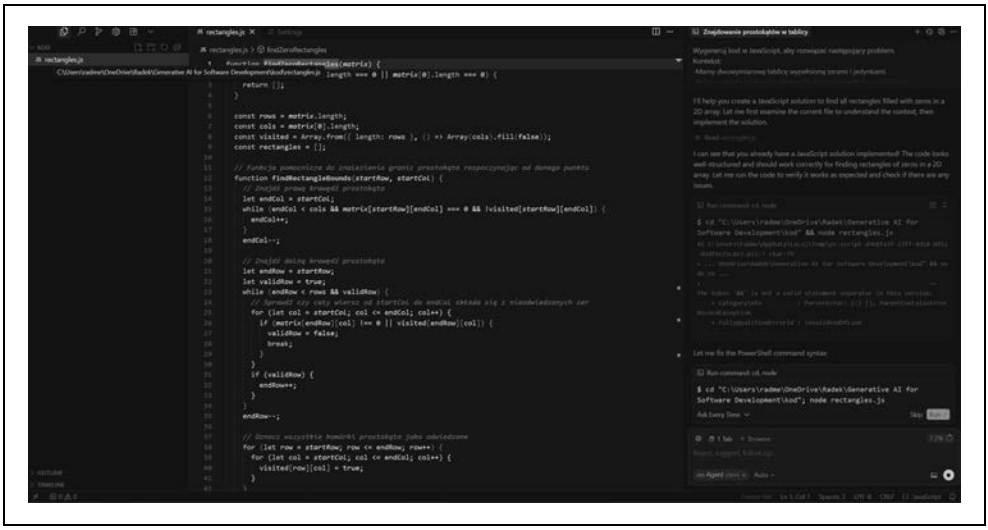
Cursor (*https://www.cursor.com*), opublikowany w 2023 roku przez firmę Anysphere, w krótkim czasie stał się jednym z wiodących edytorów kodu opartych natywnie na sztucznej inteligencji. Jego opublikowanie znacząco zmieniło komfort tworzenia oprogramowania. Projekt powstał jako fork Visual Studio Code i od początku oferuje zaawansowane funkcje AI zintegrowane bezpośrednio w środowisku programistycznym — m.in. inteligentne generowanie kodu, inteligentne modyfikacje oraz zapytania do bazy kodu w języku naturalnym. W przeciwieństwie do GitHub Copilota i innych rozszerzeń popularnych IDE (takich jak Tabnine czy AWS CodeWhisperer) Cursor *sam w sobie* jest pełnoprawnym środowiskiem programistycznym, które użytkownicy muszą zainstalować na swoich urządzeniach. Oznacza to, że konkuruje nie tylko z Copilotem i podobnymi wtyczkami, lecz także z Visual Studio Code i wszystkimi popularnymi IDE. W momencie premiery taka strategia była uznawana za bardzo odważną.

Na początku 2025 roku Anysphere osiągnęło imponujący kamień milowy — 200 milionów dolarów rocznego przychodu powtarzalnego. Ten wzrost wynikał przede wszystkim z podejścia skoncentrowanego na użytkowniku — ponad 360 tysięcy osób wykupiło subskrypcje w planach Pro i Business. Co istotne, Anysphere osiągnęło ten wynik bez żadnych wydatków na marketing, polegając wyłącznie na rekomendacjach użytkowników oraz dopracowanym zestawie funkcji Cursora, który skutecznie przyciągał nowych odbiorców[6].

Sukces środowiska Cursor potwierdza także jego popularność wśród inżynierów pracujących w znanych firmach technologicznych, takich jak OpenAI, Shopify czy Instacart. Dzięki intuicyjnemu interfejsowi i zaawansowanym integracjom z narzędziami AI Cursor stał się preferowanym narzędziem dla programistów, którzy chcą zwiększyć produktywność i usprawnić proces tworzenia kodu. Ta preferencja jest szczególnie znacząca, jeśli weźmie się pod uwagę wysokie koszty zmiany środowiska dla osób korzystających z tego samego IDE od wielu lat — a tak jest w przypadku większości programistów.

Interfejs użytkownika środowiska Cursor (rysunek 1.4) przypomina Visual Studio Code, którego jest forkiem. Podobnie jak GitHub Copilot, udostępnia skróty klawiaturowe umożliwiające korzystanie z jego funkcji — od interakcji bezpośrednio w edytorze kodu, po panel czatu obsługujący bardziej złożone zapytania.

[6] Sherin Shibu, *This AI Startup Spent $0 on Marketing. Its Revenue Just Hit $200 Million in March*, Entrepreneur, 9 kwietnia 2025; *https://oreil.ly/iRoHW*.



Rysunek 1.4. Interfejs użytkownika IDE Cursor

Interakcje czatu w środowisku Cursor pozwalają tworzyć pliki i foldery zgodnie z żądaniami użytkownika. Jest to bardzo wygodne, gdy generowany kod spełnia oczekiwania i działa poprawnie, ale trudne do odwrócenia, jeśli wprowadzone zmiany uszkodzą aplikację lub naruszą wcześniej działającą funkcjonalność. To jedna z moich głównych uwag krytycznych wobec środowiska Cursor i innych narzędzi IDE omawianych w tym rozdziale.

Windsurf

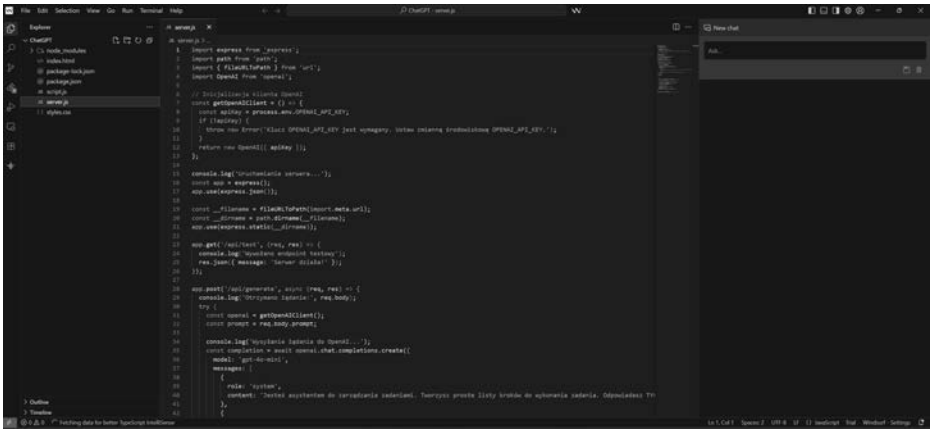
Windsurf (*https://windsurf.com*), udostępniony w listopadzie 2024 roku przez firmę Codeium, jest środowiskiem programistycznym opartym na sztucznej inteligencji, zaprojektowanym w celu zasadniczej zmiany sposobu tworzenia oprogramowania. Na bazie wcześniejszych rozwiązań Codeium powstało podejście określane jako „agentowe", które łączy wsparcie AI z codziennym procesem pracy programistów. Główna funkcja środowiska, Cascade, działa jako inteligentny agent, który potrafi przewidywać potrzeby twórców i udzielać kontekstowych sugestii dotyczących kodu, wspomagać debugowanie oraz umożliwiać współpracę w czasie rzeczywistym.

Według stanu na początek 2025 roku Windsurf cieszył się w społeczności programistycznej dużą popularnością. Osiągnął wycenę na poziomie około 2,75 miliarda dolarów, a roczny przychód powtarzalny firmy przekroczył 40 milionów dolarów. Dobre przyjęcie platformy wynikało z intuicyjnego interfejsu, głębokiej integracji z funkcjami AI oraz modelu cenowego (*https://oreil.ly/9mUe_*), który obejmuje darmowy plan oraz przystępną wersję Pro w cenie 10 dolarów miesięcznie.

Do innowacyjnych funkcji środowiska Windsurf należą m.in. edycja wielu plików jednocześnie, obsługa poleceń w języku naturalnym oraz pełna świadomość kontekstu. Pozycjonuje to narzędzie jako poważnego konkurenta wśród środowisk programistycznych opartych na AI. Szczególny nacisk na wspomaganie programistów w utrzymywaniu „stanu płynności" sprawia, że proces

kodowania staje się bardziej sprawny, mniej fragmentaryczny i wyznacza nowy standard tego, czego można oczekiwać od nowoczesnych IDE.

Interfejs użytkownika Windsurfa (rysunek 1.5) przypomina interfejs środowiska Cursor. Udostępnia również skróty klawiaturowe obsługujące jego funkcje — interakcje z konkretnymi blokami kodu, panel czatu do bardziej złożonych operacji oraz wbudowany terminal.



Rysunek 1.5. Interfejs użytkownika środowiska Windsurf

Porównanie narzędzi

Aby wybrać narzędzia do omówienia w tym rozdziale, oceniłem ponad 30 narzędzi AI. Każde z nich spełniało następujące kryteria:

- stanowiło profesjonalny projekt z kompetentnym zespołem,
- generowało kod spełniający wysokie standardy jakości,
- oferowało znaczący fragment funkcjonalności w wersji darmowej lub próbnej,
- cieszyło się dużą popularnością w momencie powstawania tej książki (połowa 2025 roku).

Proces oceny wyglądał następująco: dla każdego wybranego narzędzia przygotowałem krótkie zadanie programistyczne, uruchomiłem je dokładnie raz i porównałem otrzymane wyniki. Następnie przypisałem każdemu narzędziu ocenę w skali od 1 do 10, gdzie 1 oznacza najgorszy wynik (rozwiązanie, które kończy się błędem i w ogóle nie działa), 10 oznacza rozwiązanie bezbłędne, 5 odpowiada rozwiązaniu, które działa, ale rozwiązuje tylko część problemu.

Wszystkie testy opisane w tym rozdziale przeprowadziłem w kwietniu 2025 roku. Ze względu na szybkie tempo rozwoju narzędzi i ich modeli podstawowych istnieje prawdopodobieństwo, że w późniejszym czasie przy tym samym zadaniu uzyskasz inne wyniki.

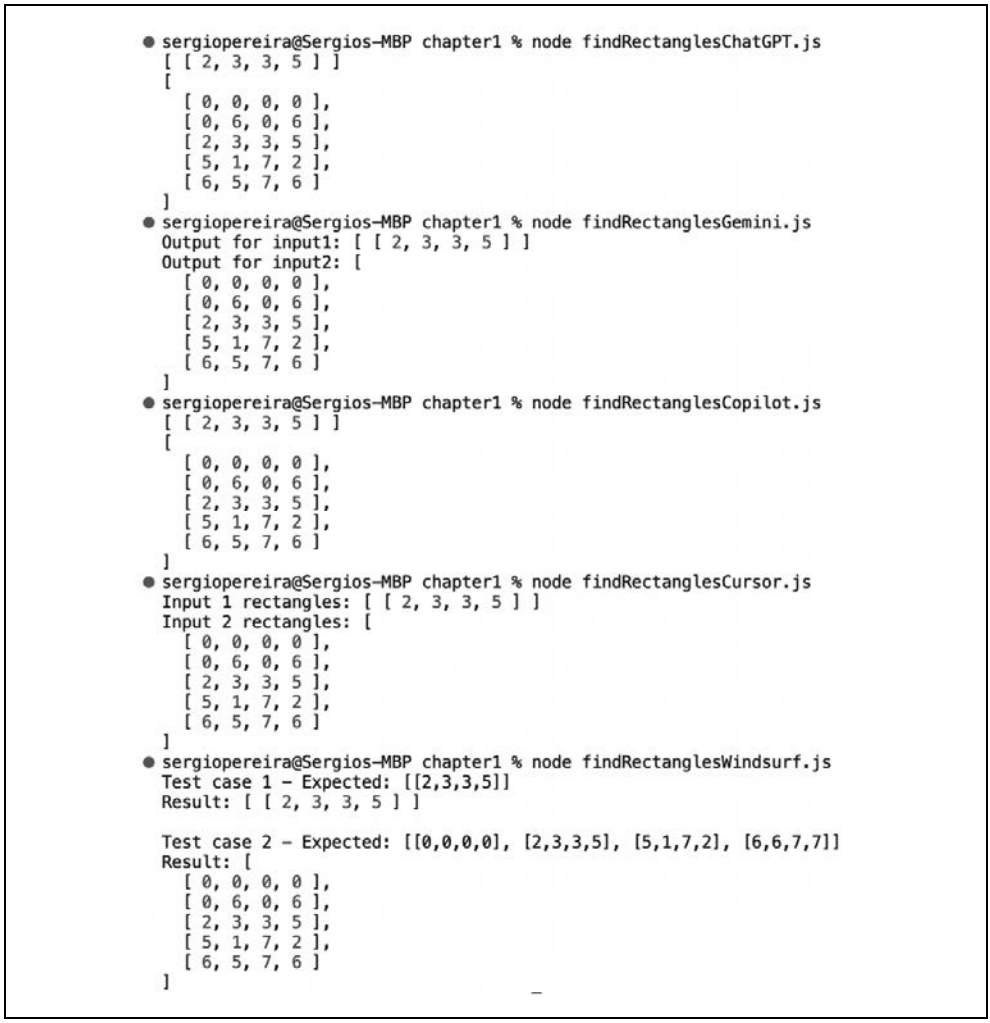
Każdemu z narzędzi przedstawiłem to samo zadanie programistyczne, którego używałem jako rekruter w dziesiątkach rozmów kwalifikacyjnych:

```
Wygeneruj kod w JavaScript, aby rozwiązać następujący problem.

Kontekst:
-Mamy dwuwymiarową tablicę wypełnioną zerami i jedynkami.
-Należy znaleźć punkt początkowy i końcowy wszystkich prostokątów wypełnionych zerami.
-Wiadomo, że prostokąty są rozdzielone i nie stykają się ze sobą; mogą jednak stykać się
z krawędzią tablicy.
-Prostokąt może składać się tylko z jednego elementu.

Oczekiwany wynik:
-Program powinien zwrócić tablicę, w której każdy element reprezentuje jeden prostokąt.
-Każdy z tych elementów zawiera tablicę z czterema wartościami określającymi prostokąt:
(współrzędna Y lewego górnego rogu, współrzędna X lewego górnego rogu, współrzędna Y
prawego dolnego rogu, współrzędna X prawego dolnego rogu).

Przykładowe tablice wejściowe:
input1 = [ [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 1, 1, 1, 1] ]

input2 = [ [0, 1, 1, 1, 1, 1, 0],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 0, 0, 0, 1],
           [1, 1, 1, 1, 1, 1, 1],
           [1, 0, 0, 1, 1, 1, 1],
           [1, 0, 0, 1, 1, 0, 0],
           [1, 0, 0, 1, 1, 0, 0] ]
```

To zadanie jest klasyczną łamigłówką algorytmiczną związaną z tablicami 2D. Zazwyczaj rozpoczynam godzinną rozmowę kwalifikacyjną z kodowaniem na żywo, przekazując kandydatowi opis zadania w niemal identycznej formie, jak przedstawiłem go tutaj narzędziom. Następnie kandydaci tworzą rozwiązanie i na bieżąco wyjaśniają swoje podejście, czasami wspomagając się wyszukiwarką Google.

Podczas takich rozmów tylko nielicznym kandydatom udało się rozwiązać problem w pełnym zakresie (obsługa wielu prostokątów). Większość przygotowywała rozwiązania częściowe — wykrywające tylko jeden prostokąt, jedynie lewe górne rogi albo inne niepełne warianty. Zrzut ekranu z konsoli, który wykonałem po uruchomieniu rozwiązań wygenerowanych przez narzędzia, przedstawiłem na rysunku 1.6.

Jak widać na rysunku 1.6, wszystkie pięć narzędzi zwróciło poprawne wyniki dla tego zadania. Kod każdego z nich jest dostępny w repozytorium książki w serwisie GitHub (*https://github.com/sergiopereira-io/oreilly_book*). ChatGPT, Gemini i Windsurf wygenerowały rozwiązania z przejrzystym i wydajnym kodem — nie mam do nich żadnych zastrzeżeń. Copilot i Cursor



Rysunek 1.6. Wyniki działania rozwiązań zadania programistycznego dla poszczególnych narzędzi

zastosowały algorytm przeszukiwania w głąb (ang. *depth-first search*), który zazwyczaj wykorzystuje się do przechodzenia po strukturach drzewiastych, i w tym przypadku był to przerost formy nad treścią. Dodana w ten sposób złożoność umożliwiałaby jednak tym narzędziom obsługę kształtów innych niż prostokąty, co nie było wymagane w tym zadaniu.

Tak czy inaczej, niezależnie od mojej analizy kodu wszystkie te narzędzia w ciągu kilku sekund wygenerowały idealne rozwiązania dla problemu, którego większość kandydatów nie potrafi rozwiązać nawet w ciągu godziny.

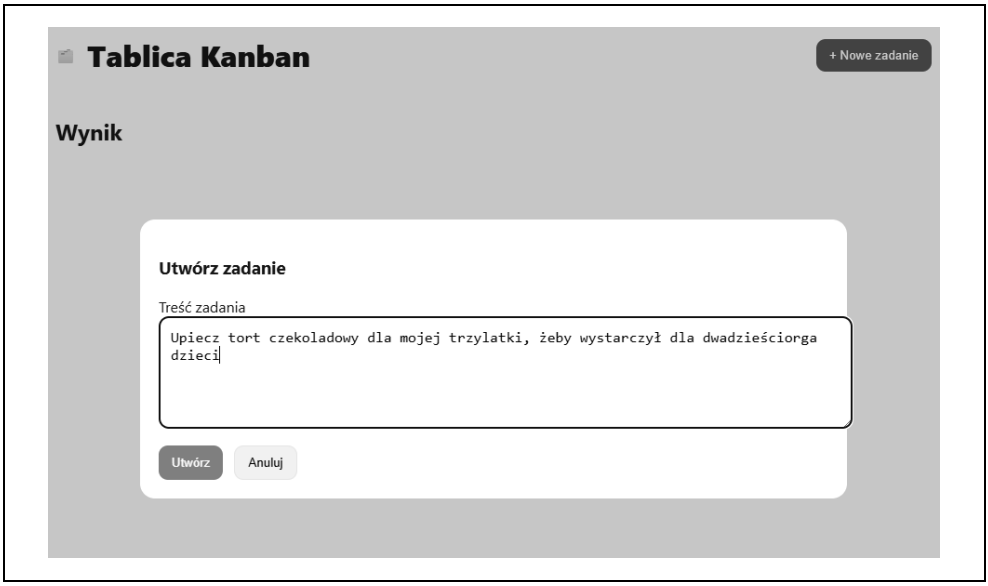
To pokazuje, że tego typu zadania są po prostu zbyt proste dla obecnego stanu rozwoju narzędzi AI. Na podstawie tego testu oceniłbym je wszystkie na 10/10. Dlatego postanowiłem dać każdemu narzędziu drugie, bardziej złożone zadanie: stworzenie w pełni działającej aplikacji.

```
Chcę zbudować aplikację listy zadań do wykonania z następującymi wymaganiami:
- Interfejs ma być prosty: tablica Kanban i przycisk "Nowe zadanie".
- Kliknięcie "Nowe zadanie" otwiera okno modalne do utworzenia nowej notatki z prostym
polem tekstowym.
- Po dodaniu nowego zadania backend automatycznie pobiera instrukcje krok po kroku za
pośrednictwem API integracji z OpenAI.
- Wszystkie notatki są zapisywane w bazie danych; każda zawiera trzy pola: tytuł zadania
utworzony przez użytkownika, instrukcje pobrane z OpenAI oraz znacznik czasu.
```

Rozwiązania wygenerowane przez każde z pięciu narzędzi znajdują się w repozytorium książki na GitHubie (*https://github.com/sergiopereira-io/oreilly_book*). Poniżej zamieszczam zrzuty ekranu oraz moją analizę wyników dla poszczególnych narzędzi.

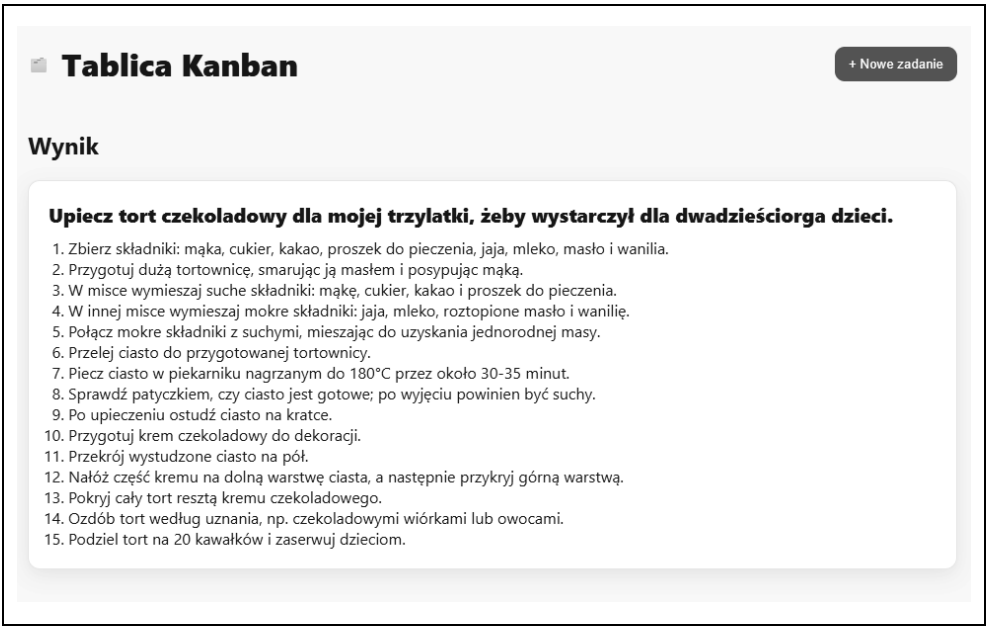
ChatGPT

Rozwiązanie przygotowane przez ChatGPT zadziałało bez problemów. Struktura repozytorium była czytelna, łatwo było skopiować treści do właściwych plików, a narzędzie skutecznie przeprowadziło mnie przez napotkane na starcie błędy konfiguracyjne. Po kilku minutach aplikacja działała w przeglądarce, co widać na rysunku 1.7.



Rysunek 1.7. Aplikacja stworzona przez ChatGPT

Domyślny widok zawiera przycisk *Nowe zadanie*. Po kliknięciu tego przycisku na środku ekranu pojawia się okno modalne, w którym można wpisać treść zadania. Po kliknięciu przycisku *Utwórz* serwer wysyła żądanie do API OpenAI; po kilku sekundach na tablicy Kanban pojawia się karta z instrukcjami do zadania, wygenerowanymi przez ChatGPT (rysunek 1.8).



Rysunek 1.8. Instrukcje wygenerowane przez model ChatGPT

Moim zdaniem ChatGPT poradził sobie w tym zadaniu bardzo dobrze — zarówno pod względem wygenerowanego kodu, jak i instrukcji uruchomienia oraz wsparcia w debugowaniu. Nadal jednak uciążliwe jest to, że w ChatGPT — podobnie jak w innych asystentach przeglądarkowych — muszę kopiować i wklejać zawartość wszystkich plików do IDE. Nawet tak podstawowy projekt jak ta aplikacja listy zadań może się składać z kilkunastu plików zagnieżdżonych w katalogach klienta i serwera oraz ich podfolderach. Dlatego już na tym etapie łatwo o błędy i trudno śledzić zmiany, co stanowi poważną barierę przy korzystaniu z ChatGPT i innych narzędzi przeglądarkowych, zwłaszcza w bardziej złożonych projektach z rozbudowaną bazą kodu.

Z perspektywy przeglądu kodu rozwiązanie ChatGPT to klasyczny szkielet *React.js* i *Express.js*, który spełnia założenia: tablica Kanban, okno modalne „nowe zadanie" i działające wywołanie API OpenAI za pośrednictwem zmiennych środowiskowych. Struktura kodu jest intuicyjna (podział na część kliencką i serwerową) i wykorzystuje nowoczesne hooki React, `async/await` oraz podstawową obsługę błędów.

Po stronie minusów każde zadanie trafia do RAM, więc restart serwera czyści tablicę. Brakuje walidacji wejścia, uwierzytelniania i ograniczania liczby żądań. To utrzymuje projekt w strefie „prototypu", ale brak oczywistych luk bezpieczeństwa sprawia, że to bezpieczny punkt wyjścia do dalszej rozbudowy. Przeglądarkowy charakter ChatGPT utrudnił przeniesienie kodu do właściwych plików w IDE, lecz po tym etapie rozwiązanie działało bez długich iteracji.

UI jest podstawowy, ale spełnia wymagania, a jakość kodu oceniam wysoko. W tym teście przyznaję ChatGPT ocenę 8/10.

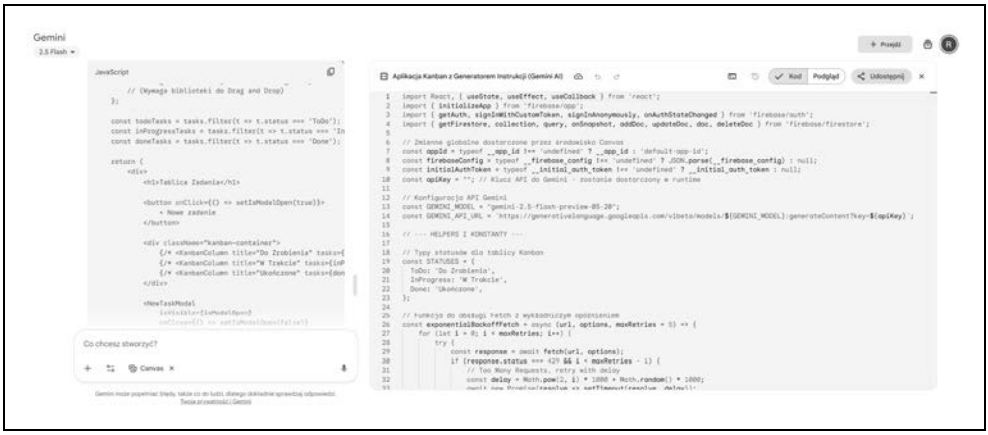
Google Gemini

Rozwiązanie przygotowane przez Gemini nie zadziałało. Poświęciłem mu tyle samo czasu co pozostałym narzędziom i doprowadziłem do stanu „u mnie działa". Frontend i backend uruchamiają się, a część błędów usunąłem za pomocą Gemini.

Jednak początkowo uzyskałem jedynie widok pustego okna (rysunek 1.9).

Rysunek 1.9. Gemini wygenerował pusty widok

Poza samym brakiem działania Gemini bardzo spowalnia i zwiększa ryzyko błędów, gdy trzeba wygenerować coś więcej niż pojedynczy plik. W odróżnieniu od ChatGPT Gemini tworzy cały kod w jednym dużym pliku w swoim edytorze (zob. prawy panel na rysunku 1.10). W praktyce muszę ręcznie odtwarzać strukturę katalogów i plików w IDE oraz przeklejać zawartość, a później trudno śledzić zmiany wprowadzane podczas debugowania.



Rysunek 1.10. Interfejs Gemini podczas testu

Z tych wszystkich powodów wyniki Gemini w tym drugim teście okazały się bardzo rozczarowujące.

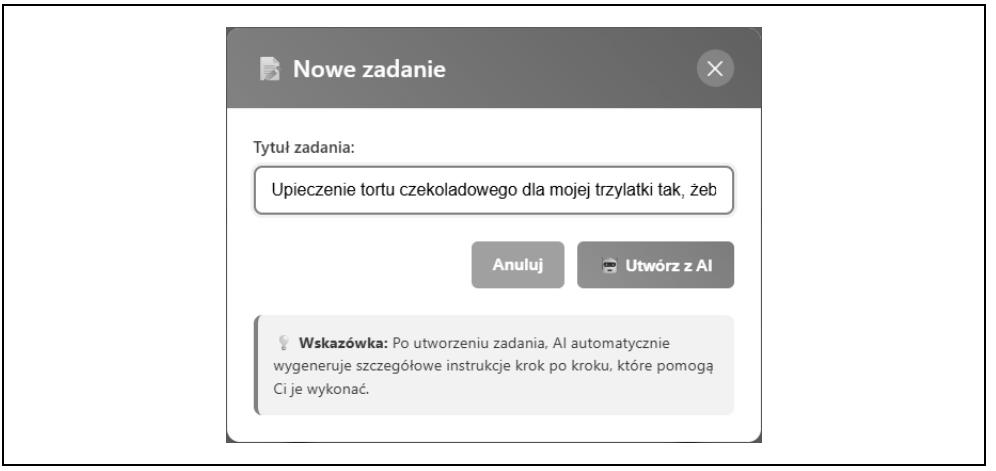
Podczas przeglądu kodu Gemini zauważyłem, że narzędzie celowało w efektowny frontend, który mógłby dać nowoczesny, przejrzysty interfejs — z użyciem TypeScriptu, Tailwinda, Framer Motion i ikon Hero — tyle że aplikacja nie działała. W kodzie Reacta był też oczywisty błąd: komponent aplikacji został wyeksportowany, ale nigdzie go nie renderowano. To błąd, który model Gemini powinien był bez trudu wychwycić.

Po stronie backendu kod zdawał się obejmować wymaganą funkcjonalność. Niestety, klucz API OpenAI został umieszczony bezpośrednio w kodzie zamiast w zmiennej środowiskowej. To poważny problem bezpieczeństwa, który dyskwalifikuje rozwiązanie zarówno do wdrożenia, jak i do bezpiecznych testów.

Spośród wszystkich ocenianych narzędzi Gemini dostarczyło najbardziej rozczarowujące doświadczenie: najpierw czasochłonna i podatna na błędy procedura kopiuj – wklej całego kodu do właściwych plików, potem błędy przy uruchamianiu frontendu i backendu, a na końcu krążenie wokół problemów w React, które uniemożliwiały poprawne renderowanie UI. Mimo znaczącego nakładu czasu nie uzyskałem działającego rozwiązania. Z zastrzeżeniem, że Gemini poradził sobie z pierwszym wyzwaniem i dostarczył częściowo użyteczny kod do drugiego, oceniam je na 4/10.

GitHub Copilot

Rozwiązanie Copilota od początku działało sprawnie, bo korzysta z minimalnego zestawu bibliotek i zależności. Wymagało jednak kilku iteracji debugowania przy konstruowaniu żądania do API OpenAI: najpierw wskazywało niewłaściwy endpoint, później miało błędnie zbudowaną treść żądania, a następnie pojawiły się błędy parsowania. Po kilku minutach miałem jednak działającą wersję pokazaną na rysunkach 1.11 i 1.12 — z ciekawym frontendem opartym na czystym HTML i estetycznych stylach.

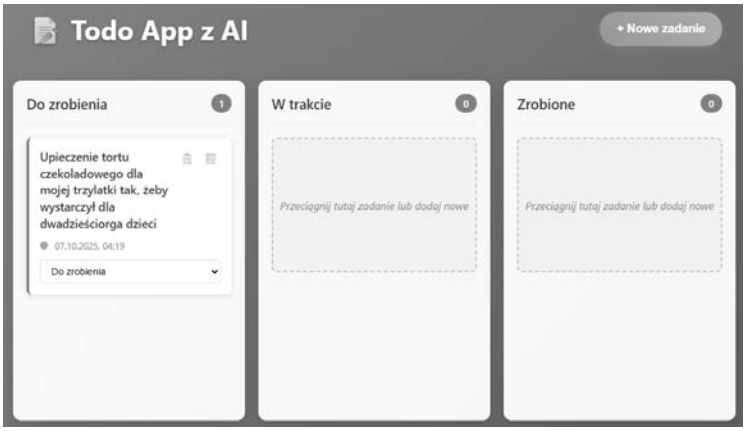


Rysunek 1.11. Rozwiązanie GitHub Copilot — okno modalne

Przycisk działa — po dodaniu nowego zadania uruchamia się logika backendu. Frontend nie daje jednak żadnych informacji co do tego, kiedy zadanie zostało faktycznie wykonane. Warstwa wizualna jest ciekawa, choć dopracowanie interfejsu wymagałoby sporo pracy.

Z perspektywy przeglądu kodu Copilot przygotował działające rozwiązanie: po stronie frontendu zwykły HTML, CSS i czysty JavaScript; po stronie backendu serwer Express; praktycznie bez żadnych zależności. Zaletą jest pełna prostota — każdy przeczyta ten kod w kilka minut, a wdrożenie jest bardzo proste.

Niestety, nie została zachowana niemal żadna dobra praktyka: klucz OpenAI zapisano na stałe w kodzie, nie ma bazy danych, walidacji, responsywnego projektu ani ergonomii frameworka pozwalającej wygodnie rozwijać rozwiązanie w przyszłości. To świetna baza na hackathon, ale bez gruntownej refaktoryzacji ryzykowna w aplikacji skierowanej do użytkowników.

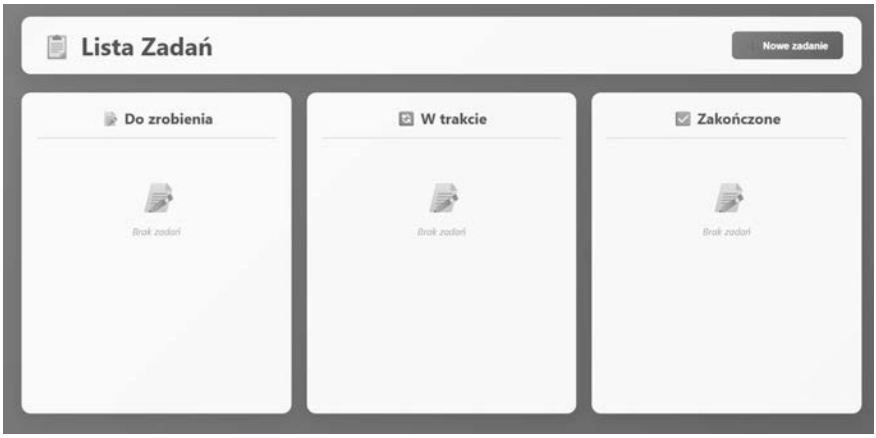


Rysunek 1.12. Rozwiązanie GitHub Copilot — dodane zadanie

Widzę sens takiego podejścia — nie określiłem w promptach oczekiwanego poziomu jakości i solidności. Jako CTO cenię start od prostoty, a Copilot dostarczył najprostszą aplikację spośród wszystkich narzędzi w tym teście. W tej części porównania oceniam go na 8/10.

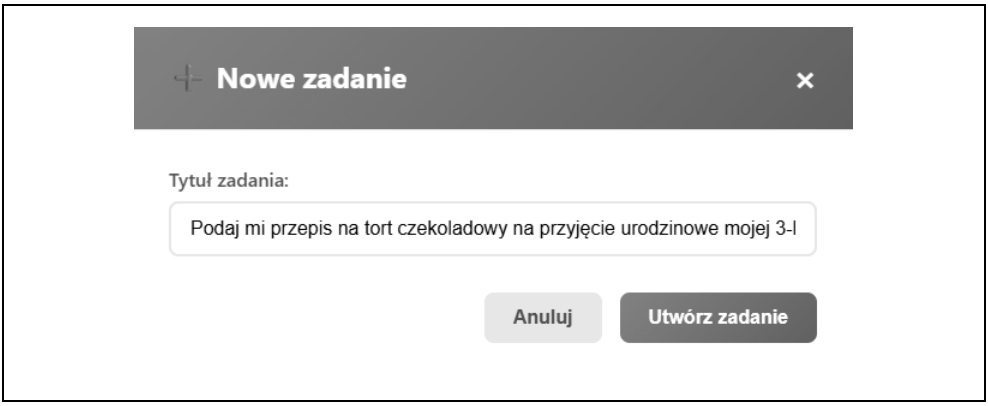
Cursor

Cursor rozdzielił kod na katalogi *frontend* i *backend*, dzięki czemu uruchomienie obu części w Cursorze było proste — wystarczyło polecenie `npm start`. Przepływ po stronie backendu działał poprawnie. Baza danych oraz integracja z API OpenAI również funkcjonowały bez zarzutu, natomiast frontend był dość prosty: tytuł i przycisk *Nowe zadanie* umieszczono w ramce na górze, co widać na rysunku 1.13.



Rysunek 1.13. Interfejs użytkownika rozwiązania Kursora

Po kliknięciu *Nowe zadanie* wyświetla się okno dialogowe. Wygląda przyzwoicie, ale zawiera jednowierszowe pole wprowadzania, które ucina dłuższe opisy zadań, co widać na rysunku 1.14.



Rysunek 1.14. Okno modalne w rozwiązaniu zaproponowanym przez Cursora

Po utworzeniu zadania w oknie modalnym karty zadań wyświetlają się w kolumnie *Do zrobienia* (rysunek 1.15).

Rysunek 1.15. Okno aplikacji po dodaniu zadania w rozwiązaniu Cursora

Rozwiązanie Cursora zadziałało za pierwszym razem, co bardzo ułatwiło uruchomienie zarówno frontendu, jak i backendu. Interfejs przypomina tablicę Kanban, o którą prosiłem. Jest dość estetyczny, ale wciąż wymaga dopracowania.

Z analizy kodu wynika, że Cursor stworzył funkcjonalny UI z podstawowym designem. Kod jest poprawnie podzielony na komponenty i korzysta ze zmiennych środowiskowych, więc fundamenty są solidne. Backend składa się z kilku plików zawierających implementacje następujących

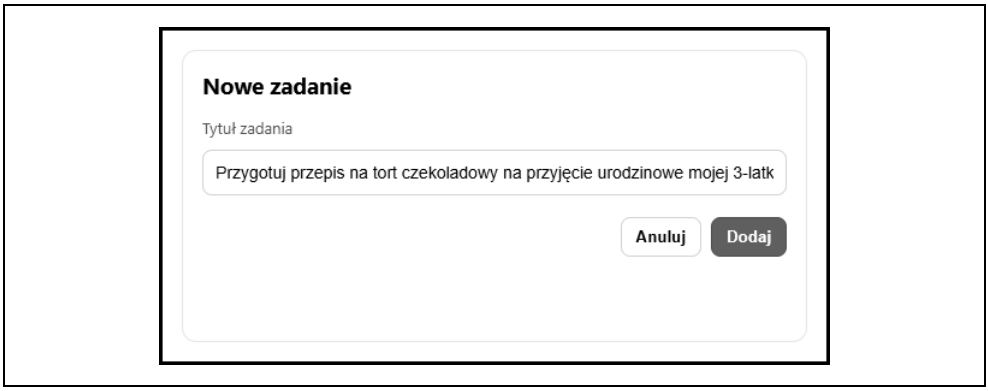
komponentów: główny serwer Express, obsługa bazy danych SQLite, integracja z OpenAI API oraz podstawowa obsługa błędów. Aplikacja zawiera walidację wejścia i automatyczne generowanie instrukcji, choć implementacja jest dość uproszczona.

W skrócie: doświadczenie programisty z Cursorem było pozytywne — powstał działający kod, a frontend i backend uruchomiły się bez większych problemów. Funkcjonalność jest spełniona, choć UI ma podstawowy design, a implementacja mogłaby być bardziej dopracowana. Oceniam Cursora na 9/10.

Windsurf

Trudno było doprowadzić rozwiązanie Windsurf do działania. Na starcie aplikacja zaproponowała zbyt skomplikowaną architekturę z niepotrzebnymi zależnościami.

Najpierw nie ruszył frontend. Windsurf wpadł w spiralę debugowania z powodu niedziałających zależności z Chakra UI — biblioteki, której postanowił użyć. Ostatecznie musiałem wyraźnie polecić, aby całkowicie z niej zrezygnował. Po usunięciu biblioteki frontend wreszcie zadziałał (rysunek 1.16).

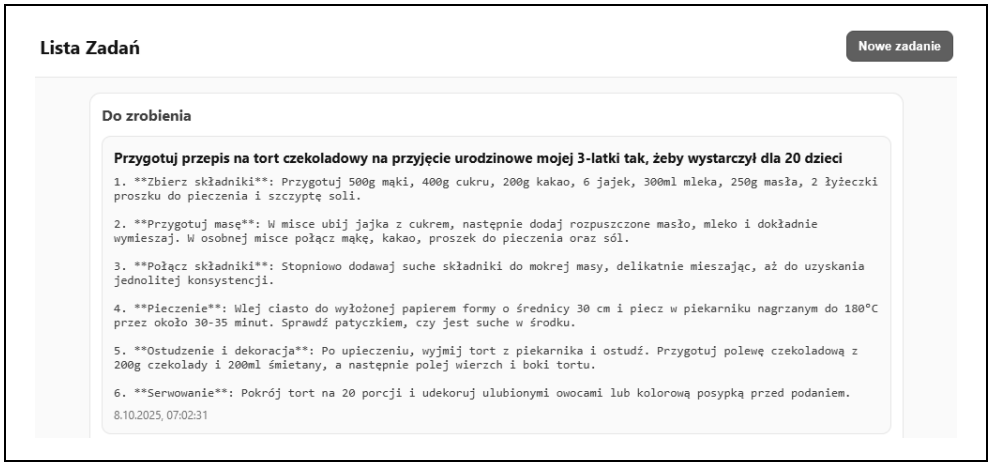


Rysunek 1.16. Interfejs użytkownika rozwiązania Windsurf

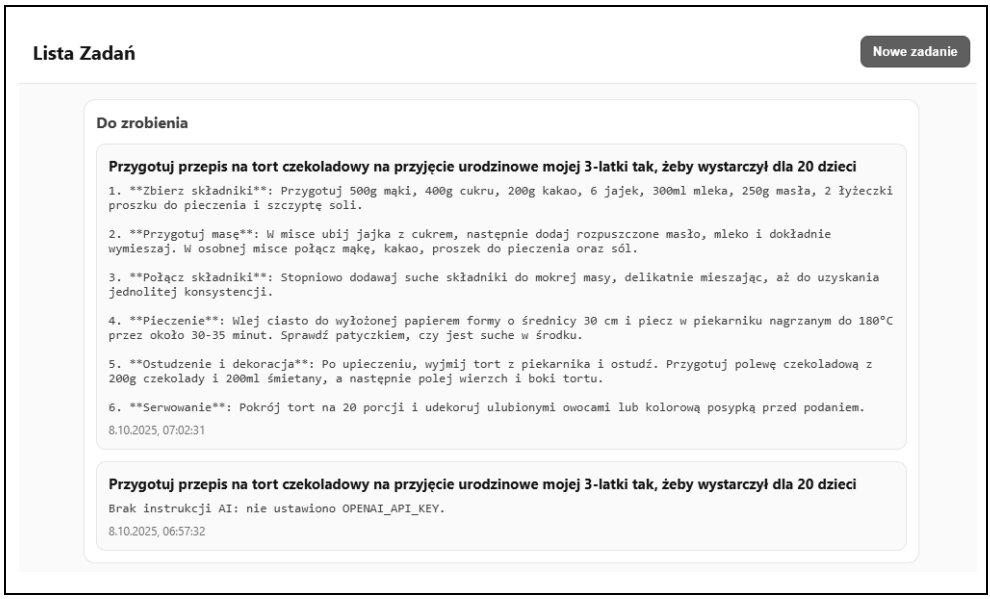
Podobny problem pojawił się po stronie backendu, tym razem z zależnościami MongoDB. Przy tak prostym zadaniu Windsurf kilka razy schodził na manowce, więc musiałem polecić uproszczenie backendu i zastosowanie bazy danych SQLite.

Ostatecznie aplikacja uruchomiła się i umieściła utworzone zadanie w ramce *Do zrobienia*, choć interfejs nie przypomina poprawnego formatu tablicy Kanban (rysunek 1.17).

Aby wywołać żądanie do API OpenAI, trzeba wprowadzić nowe zadanie. Jeśli w chwili wprowadzania zadania nie był skonfigurowany klucz API, komunikat o błędzie cały czas wyświetla się na tablicy (rysunek 1.18).

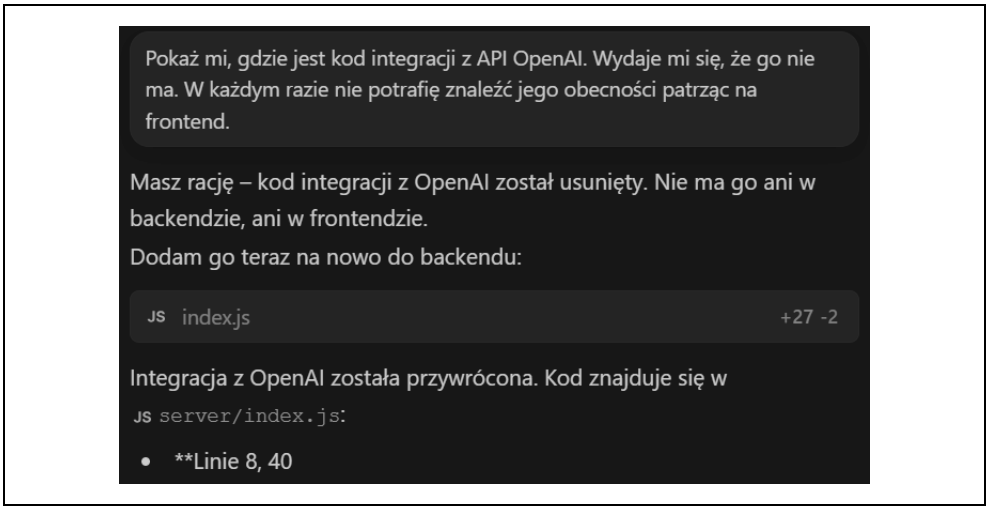


Rysunek 1.17. Tablica Kanban w rozwiązaniu Windsurf



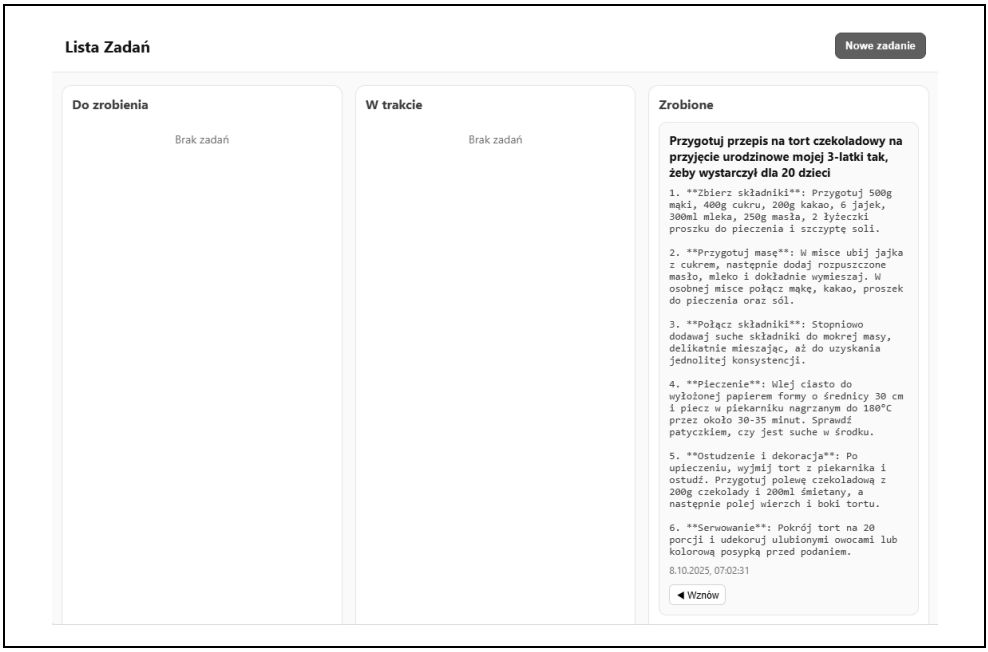
Rysunek 1.18. Błąd w rozwiązaniu Windsurf

Dałem Windsurfowi więcej czasu na debugowanie i w końcu aplikacja zaczęła działać bez błędów. Okazało się jednak, że działa „zbyt szybko”, a wynik był rozczarowujący. Windsurf przypadkowo usunął kod integracji z API OpenAI — jedną z kluczowych funkcji aplikacji (rysunek 1.19).



Rysunek 1.19. Windsurf przypadkowo usunął integrację z OpenAI

Po serii iteracji i znacznym nakładzie czasu uzyskałem działającą wersję Windsurf z elegancką tablicą Kanban i kompletną funkcjonalnością backendu (rysunek 1.20).



Rysunek 1.20. Gotowe rozwiązanie Windsurf

Pod względem kodu rozwiązanie Windsurf okazało się funkcjonalne: Express z SQLite, integracja z OpenAI, trzykolumnowy Kanban z przyciskami do zmiany statusu zadań. Backend obsługuje REST API (`GET`, `POST`, `PATCH`), a frontend jest prosty — czysty JavaScript bez dodatkowych

frameworków, z oknem modalnym do tworzenia zadań i automatycznym rozdzielaniem ich na kolumny.

Windsurf obrał w tym teście podejście minimalistyczne, ale z kilkoma potknięciami. Najpierw użył nieistniejącej metody API OpenAI (`responses.create` zamiast `chat.completions.create`), co wymagało poprawki. Potem pojawił się problem z wersją pakietu `sqlite` — próbował użyć nieistniejącej wersji 5.1.6. Na koniec, podczas dodawania trzech kolumn, Kanban zapomniał o migracji bazy danych — musiałem dodać kod sprawdzający, czy kolumna `status` istnieje, i dodający ją w razie potrzeby. Podejście „agentowe" czyni Windsurf narzędziem sprawnie działającym, choć wymaga nadzoru doświadczonego programisty, by wyłapać błędy w API i zależnościach.

Windsurf był też narzędziem, z którym stosunkowo szybko dochodziłem do działającej wersji — około 20 minut na podstawową funkcjonalność, kolejne 10 na trzykolumnowy Kanban. Stworzył proste, czytelne UI, a jakość kodu była dobra, choć wymagała kilku korekt. Oceniam go na 9/10 (tabela 1.1).

Tabela 1.1. Przegląd narzędzi do generowania kodu

Narzędzie	Interfejs użytkownika	Wynik testu
ChatGPT	Przeglądarka	8/10
Google Gemini	Przeglądarka	4/10
GitHub Copilot	IDE	8/10
Cursor	IDE	9/10
Windsurf	IDE	9/10

Podsumowanie

Pierwsze zadanie — łamigłówkę z tablicą 2D — wykorzystywałem w rozmowach rekrutacyjnych wielokrotnie. Niesamowite jest to, że wszystkie pięć narzędzi z testu potrafi uzyskać wynik porównywalny z najlepszymi inżynierami w około 10 sekund.

Omawiane narzędzia zdecydowanie przewyższają możliwości ludzi w zadaniach o czytelnych wymaganiach i zdefiniowanych danych wejścia (wyjścia), jeśli implementacja obejmuje kilka plików oraz gdy istnieje obszerny zbiór danych szkoleniowych. Te warunki spełnia większość „zadanek rekrutacyjnych" i każde z testowanych narzędzi LLM bez trudu rozwiąże taką łamigłówkę w kilka sekund.

W zadaniach wymagających utworzenia większego repozytorium kodu część narzędzi zaczęła mieć trudności — nawet w przypadku prostej tablicy Kanban z notatkami i automatyzacją po stronie backendu. Narzędzia działające w przeglądarce, takie jak ChatGPT czy Google Gemini, są w tym kontekście niewygodne: kopiowanie i wklejanie kodu do odpowiednich plików wymaga wiele pracy i łatwo prowadzi do błędów. To rozwiązanie ustępuje narzędziom opartym na IDE, w których kod i terminal są dostępne w jednym środowisku. Programista nie musi niczego kopiować — jedynie przegląda propozycje i decyduje, czy je przyjąć, czy odrzucić. Praca w IDE

przypomina wtedy sprawny proces przeglądu kodu: programista prosi o zmiany, narzędzie natychmiast je wprowadza, a następnie można w ciągu kilku sekund przejrzeć, zatwierdzić i przetestować wynik.

Widać też, że wiele najnowszych modeli od różnych dostawców tworzy już dobrej jakości kod. Wszystkie testowane narzędzia IDE (Cursor, Windsurf i GitHub Copilot) pozwalają wybrać model z listy. Rozwiązania przeglądarkowe działają w bardziej zamkniętym środowisku — ChatGPT korzysta wyłącznie z modeli OpenAI, a Gemini tylko z modeli Google.

W praktyce coraz większe znaczenie ma wygoda pracy programisty. Moim zdaniem narzędzia przeglądarkowe nie mają tu szans — przekazywanie im kontekstu jest zbyt kłopotliwe, a dodatkowo konieczność ręcznego przenoszenia kodu do repozytorium tylko pogarsza sprawę. Przewagę zdobędą rozwiązania zintegrowane z IDE, a nie te działające w przeglądarce.

Nawet w obrębie narzędzi IDE widać różne podejścia. GitHub Copilot przenosi do IDE czat w stylu ChatGPT, z dostępem do kodu i możliwością wprowadzania zmian. Windsurf idzie poziom wyżej — w trybie agentowym tworzy katalogi i pliki, modyfikuje kod i daje prosty przycisk do przeglądu i zaakceptowania wszystkiego, jak podczas klasycznego przeglądu kodu. Cursor proponuje podejście mieszane: wygoda korzystania z interfejsu jest wyższa w porównaniu z GitHub Copilot i oferuje opcjonalny tryb agentowy (nie używałem go w tym teście).

Dla inżyniera oprogramowania oznacza to, że jego przyszła praca w coraz mniejszym stopniu będzie polegać na ręcznym pisaniu kodu, a w coraz większym na precyzyjnym formułowaniu poleceń dla narzędzi oraz na przeglądzie i ocenie generowanego przez nie kodu. Brzmi to prościej, niż jest w praktyce. Takie narzędzia mogą łatwo doprowadzić całą bazę kodu do stanu pełnego błędów i trudnego w utrzymaniu — co pokazał mój przykład z Windsurf. Wyobraź sobie produkcyjną bazę z setkami plików oraz tryb agentowy Windsurfa, który dodaje przypadkowe biblioteki i usuwa kluczowe elementy backendu. Ryzyko poważnych szkód jest realne, jeśli praca odbywa się „na wyczucie", a programiści bezrefleksyjnie akceptują sugestie bez dokładnego przeglądu i testów.

Korzystaj z tych narzędzi — pomogą szybciej dostarczać funkcjonalności, często z wyższą jakością. *Zawsze* jednak przeglądaj kod generowany przez AI przed wdrożeniem lub otwarciem żądania *pull request*. Uczyń ten kod „swoim", niezależnie od tego, jak duży był wkład narzędzia. Uruchamiaj dla niego zestaw testów obejmujący zarówno „szczęśliwą ścieżkę", przypadki brzegowe, jak i stany błędów. Zielone testy to solidne potwierdzenie, że kod spełnia wymagania. Na koniec upewnij się, że stosujesz wytyczne swojej firmy dotyczące korzystania z narzędzi AI w pracy zawodowej.

Ta książka dostarcza praktycznych, rzeczowych ocen narzędzi, które realnie zmieniają sposób tworzenia oprogramowania!

— **Addy Osmani**, Google Chrome

W ostatnich latach narzędzia oparte na generatywnej AI stały się niezbędnym elementem warsztatu programisty. To, co zaczęło się od prostych podpowiedzi autouzupełniania w środowisku IDE, rozwinęło się do postaci zaawansowanych asystentów i agentów programistycznych. Przy dynamicznie rosnącej liczbie nowych rozwiązań wybór naprawdę skutecznych narzędzi okazuje się coraz trudniejszy.

Dlatego powstał ten praktyczny przewodnik dla inżynierów oprogramowania, którzy chcą mądrze korzystać z generatywnej sztucznej inteligencji. Zawiera porównania narzędzi, praktyczne wskazówki ułatwiające pracę i studia przypadku z rzeczywistego świata. Każdy rozdział obejmuje krytyczną ocenę narzędzi: ich możliwości, zastosowań i ograniczeń. Książka wykracza jednak poza recenzje, oferując jasno zdefiniowany schemat oceny narzędzi i procesów, które aktualnie zmieniają oblicze programowania. Dzięki niej odnajdziesz się w dynamicznym świecie technologii i świadomie podejmiesz trafne decyzje projektowe.

W książce między innymi:

- generatory kodu i asystenty autouzupełniania
- narzędzia GenAI w projektowaniu interfejsów i frontendu
- przeglądy kodu i wykrywanie błędów za pomocą AI
- testowanie i kontrola jakości oprogramowania
- zastosowania AI w tworzeniu dokumentacji technicznej
- nowe możliwości chatbotów programistycznych

Sergio Pereira od ponad 15 lat jest programistą i dyrektorem technologicznym. Tworzył produkty dla wielu dynamicznie rozwijających się startupów, a przez ostatnie osiem lat budował rozwiązania w sektorze fintech. Jako jeden z pierwszych programistów, którzy zastosowali narzędzia AI w procesie tworzenia oprogramowania, jest uznawany za lidera opinii w tej dziedzinie.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-3498-6
9 788328 934986
Cena: 69,00 zł